Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: Programming Productivity, Performance, and Energy Consumption*

Suejb Memeti
Linnaeus University
Växjö, Sweden 351 95
suejb.memeti@lnu.se

Lu Li
Linköping University
Linköping, Sweden 581 83
lu.li@liu.se

Sabri Pllana
Linnaeus University
Växjö, Sweden 351 95
sabri.pllana@lnu.se

Joanna Kołodziej
NASK
Warsaw, Poland 01 045
jokolodziej@pk.edu.pl

Christoph Kessler
Linköping University
Linköping, Sweden 581 83
christoph.kessler@liu.se

ABSTRACT

Many modern parallel computing systems are heterogeneous at their node level. Such nodes may comprise general purpose CPUs and accelerators (such as, GPU, or Intel Xeon Phi) that provide high performance with suitable energy-consumption characteristics. However, exploiting the available performance of heterogeneous architectures may be challenging. There are various parallel programming frameworks (such as, OpenMP, OpenCL, OpenACC, CUDA) and selecting the one that is suitable for a target context is not straightforward. In this paper, we study empirically the characteristics of OpenMP, OpenACC, OpenCL, and CUDA with respect to programming productivity, performance, and energy. To evaluate the programming productivity we use our homegrown tool CodeStat, which enables us to determine the percentage of code lines required to parallelize the code using a specific framework. We use our tools MeterPU and x-MeterPU to evaluate the energy consumption and the performance. Experiments are conducted using the industry-standard SPEC benchmark suite and the Rodinia benchmark suite for accelerated computing on heterogeneous systems that combine Intel Xeon E5 Processors with a GPU accelerator or an Intel Xeon Phi co-processor.

CCS CONCEPTS

• **Computing methodologies** → **Parallel programming languages**; • **Computer systems organization** → *Heterogeneous (hybrid) systems*; • **Hardware** → *Power and energy*;

KEYWORDS

OpenCL; OpenACC; OpenMP; CUDA; Programming productivity; Performance; Energy consumption

1 INTRODUCTION

Modern parallel computing systems may comprise multi-core and many-core processors at their node level. Multi-core processors may have two or more cores, and usually run at a higher frequency than many-core processors. While multi-core processors are suitable for general-purpose tasks, many-core processors (such as the Intel Xeon Phi [4] or GPU [13]) comprise a larger number of lower frequency cores that perform well on specific tasks.

Due to their different characteristics, engineers often combine multi-core and many-core processors to create the so-called heterogeneous nodes that may, if carefully utilized, result in high performance and energy efficiency. Yan et al. [21] highlight the importance of efficient node-level execution of parallel programs also for future large-scale computing systems [1, 5]. However, utilizing the available resources of these systems to the highest possible extent require advanced knowledge of vastly different parallel computing architectures and programming frameworks [10, 19].

Some of the widely used parallel programming frameworks for heterogeneous systems include OpenACC [20], OpenCL [17], OpenMP [14], and NVIDIA CUDA [12]. The challenge for the program developer is to choose one from many available parallel programming frameworks that fulfills in the specific context the goals with respect to programming productivity, performance, and energy consumption. Existing work has systematically studied the literature about GPU-accelerated systems [11], compared programming productivity of OpenACC and CUDA using a group of students to parallelize sequential codes [8], or used kernels for comparing execution time of OpenCL and CUDA-based implementations.

In this paper, we benchmark four well-known programming frameworks for heterogeneous systems: OpenMP, OpenACC, OpenCL, and CUDA. In addition to the industry-standard benchmark suite SPEC Accel [6], we use the popular Rodinia [3] benchmark suite to evaluate programing productivity, energy efficiency, and performance. We use our tool developed for this study *CodeStat* to quantify the programming effort for parallelizing benchmark suites

*This article is based upon work from COST Action IC1406 High-Performance Modelling and Simulation for Big Data Applications (cHiPSet), supported by COST (European Cooperation in Science and Technology).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ARMS-CC'17, July 28, 2017, Washington, DC, USA.
© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.
978-1-4503-5116-4/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3110355.3110356

under study. Furthermore, we developed *x-MeterPU* that is an extension of MeterPU, which enables us to measure the performance and energy consumption on systems that are accelerated with the Intel Xeon Phi and GPU. We present and discuss results obtained on two heterogeneous computing systems: *Emil* that comprises two Intel Xeon E5 processors and one Intel Xeon Phi co-processor, and *Ida* that has two Intel Xeon E5 processors and one GTX Titan X GPU.

The major contributions of this paper include (1) development of measurement tools CodeStat and x-MeterPU for accelerated systems with Intel Xeon Phi and GPU; (2) empirical study of four widely used frameworks for programming heterogeneous systems OpenCL, OpenACC, OpenMP, and CUDA; (3) joint consideration of programming productivity, performance, and energy consumption.

The rest of this paper is structured as follows. Section 2 describes our approach, tools and infrastructure for benchmarking programs developed with OpenMP, OpenCL, OpenACC, and CUDA. Results of the experimental evaluation using SPEC and Rodinia benchmark suites are described in Section 3. Section 4 discusses the related work. We conclude the paper in Section 5.

2 OUR METHODOLOGY AND TOOLS

In this section, we describe our approach and tools that we have developed for benchmarking parallel frameworks for heterogeneous systems. Our approach is based on using industry-standard benchmark suites for evaluation of parallelization effort, performance, and energy consumption. Figure 1 depicts our infrastructure for evaluation of parallel programming languages for accelerated systems that includes measurement tools CodeStat and x-MeterPU, benchmark suites Rodinia and SPEC Accel, and heterogeneous systems *Ida* and *Emil*.

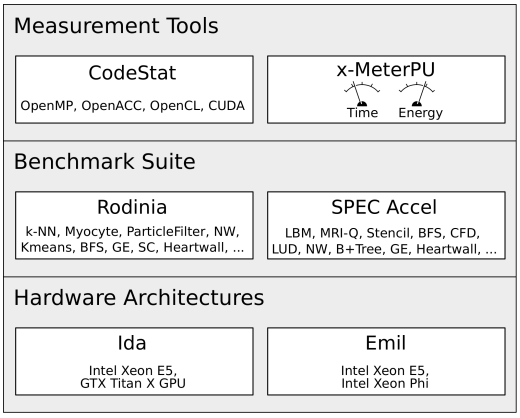


Figure 1: An overview of our infrastructure for evaluation of parallel programming languages for accelerated systems.

Table 1 summarizes major features of parallel languages that are considered in our study: OpenMP [14], OpenACC [20], OpenCL [17], and CUDA [12]. OpenMP and OpenACC are largely implemented as *compiler directives* for C, C++, and FORTRAN, which significantly hide architecture details from the programmer. OpenCL and CUDA are implemented as software libraries for C and C++ and expose the programmer to low-level architectural details. With respect to the parallelism support, all of the considered frameworks support data parallelism and asynchronous task parallelism. While OpenMP and OpenCL provide parallelization patterns for both host multi-core CPUs and many-core accelerated devices, OpenACC and CUDA support parallelization means only for accelerators such as NVIDIA GPUs [21].

Table 1: Major features of OpenACC, OpenMP, OpenCL, and CUDA.

	OpenACC	OpenMP	OpenCL	CUDA
Parallelism	- data parallelism - asynchronous task parallelism - device only	- data parallelism - asynchronous task parallelism - host and device	- data parallelism - asynchronous task parallelism - host and device	- data parallelism - asynchronous task parallelism - device only
Architecture abstraction	- memory hierarchy - explicit data mapping and movement	- memory hierarchy - data and computation binding - explicit data mapping and movement	- memory hierarchy - explicit data mapping and movement	- memory hierarchy - explicit data mapping and movement
Synchronization	- reduction - join	- barrier - reduction - join	- barrier; - reduction	- barrier
Framework implementation	compiler directives for C/C++ and Fortran	compiler directives for C/C++ and Fortran	C/C++ extensions	C/C++ extensions

2.1 CodeStat - A tool for quantifying the parallelization effort

To quantify the programming effort required to parallelize a code, we have developed our tool, named *CodeStat* [2]. *CodeStat* takes as input a configuration file and the source code. The configuration file contains a list of valid file extensions (such as, *.c, .cpp, .cu, .cl,* ...), and a list of framework specific method calls, or compiler directives (such as, *#pragma omp, proc_bind,* or *omp_set_num_threads* in OpenMP).

CodeStat analyzes the code by looking for framework specific code-statements provided in the configuration file, and provides as output the total number of lines of code (LOC) and the number of LOC written in OpenMP, OpenCL, OpenACC, or CUDA.

The configuration files for OpenMP, OpenCL, OpenACC, and CUDA are provided by *CodeStat*. Other programming frameworks can be supported by simply creating a new configuration file and adding the framework specific code-statements to the list. The list of statements for a given programming language is usually provided in the corresponding documentation or reference guides.

2.2 x-MeterPU - A tool for performance and energy consumption measurement

To measure the execution time and the energy consumption of the GPU accelerated systems we use MeterPU [9]. MeterPU is a C++ software package that implements its generic yet simple measurement API. MeterPU allows easy measurement for different metrics (e.g., time, energy) on different hardware components (e.g. CPU, DRAM, GPU). Utilizing the template and meta-programming

[2] https://github.com/suejb/CodeStat

features of C++, MeterPU's overhead is quite low and unnoticeable. It enables easy modeling and tuning for energy, besides it also allows for smooth transition for legacy tuning software (e.g. SkePU[9]) from time optimization to energy optimization if some assumptions used for time modeling are not violated for energy modeling.

To measure the execution time and the energy consumption of the Intel Xeon Phi, we have developed a variant named *x-MeterPU* [3]. x-MeterPU supports energy measurements for both native and offload programming models of Intel Xeon Phi. It is able to automatically detect the execution environment, therefore a single API function is used for measurements for both native and offload applications. To measure the energy of offload-based applications, we use the *micsmc* utility, whereas the *micras* tool is used to measure the energy of the native-based applications.

The use of x-MeterPU is very simple. The *start()* and *stop()* methods are used to enclose code regions to be measured. The *get_value()* method is used to retrieve the energy consumption (in Joules). In addition to the total energy consumption, x-MeterPU returns a log file containing all the power data with exact timestamps, which enables the production of various plots.

3 EVALUATION

In this section, we experimentally evaluate the selected parallel programming frameworks using various benchmark applications and architectures. We describe (1): the experimentation environment, including hardware configuration, benchmark applications, and evaluation metrics; and (2) the comparison results of OpenMP, OpenACC, OpenCL, and CUDA with respect to programming productivity, performance, and energy consumption.

3.1 Experimentation Environment

In this section, we describe the experimentation environment used to evaluate the selected parallel programming frameworks on heterogeneous systems. We describe the hardware configuration, the considered benchmark applications, and the evaluation metrics.

3.1.1 Hardware Configuration. For experimental evaluation of the selected parallel programming frameworks, we use two heterogeneous single-node systems.

Emil is a heterogeneous system that consists of two Intel Xeon E5-2695 v2 general purpose CPUs on the host, and one Intel Xeon Phi 7120P co-processing device. In total, the host is composed of 24 cores, each CPU has 12 cores that support two threads per core (known as logical cores) that amounts to a total of 48 threads. The Xeon Phi device has 61 cores running at 1.2 GHz base frequency, four hardware threads per core, which amounts to a total of 244 threads. One of the cores is used by the lightweight Linux operating system installed on the device.

Ida is a heterogeneous system that consists of two Intel Xeon E5-2650 v4 general purpose CPUs on the host, and one GeForce GTX Titan X GPU. Similar to *Emil*, *Ida* has 24 cores and 48 threads on the host, whereas the GPU device has 24 Streaming Multiprocessors (SM), and in total 3072 CUDA cores running at base frequency of 1 GHz. The major features of *Emil* and *Ida* are listed in table 2.

[3] https://github.com/suejb/x-MeterPU

Table 2: The system configuration details for *Emil* and *Ida*.

	Ida		Emil	
Specs	Intel Xeon E5	GeForce GPU	Intel Xeon E5	Intel Xeon Phi
Type	E5-2650 v4	GTX Titan X	E5-2695 v2	7120P
Core Frequency	2.2 - 2.9 GHz	1 - 1.1 GHz	2.4 - 3.2 GHz	1.2 - 1.3 GHz
# of Cores	12	3072	12	61
# of Threads	24	/	24	244
Cache	30 MB	/	30 MB	30.5 MB
Mem. Bandwidth	76.8 GB/s	336.5 GB/s	59.7 GB/s	352 GB/s
Memory	384 GB	12 GB	128 GB	16 GB
TDP	105 W	250 W	115 W	300 W

3.1.2 Benchmark Applications. In this paper we have considered a number of different applications from the SPEC Accel [16] and Rodinia [15] benchmark suites. The Standard Performance Evaluation Corporation (SPEC) Accel benchmark suite focuses on the performance of compute intensive parallel computing applications using accelerated systems. It provides a number of applications for OpenCL and OpenACC. Similarly, the Rodinia benchmark suite provides a number of different applications for OpenMP, OpenCL, and CUDA.

While SPEC Accel provides in total 19 OpenCL and 15 OpenACC applications, we have selected only 14 OpenCL and 3 OpenACC applications, whereas Rodinia provides 25 applications, however we have selected only 19 of them to use for experimentation (see table 3). The inclusion criteria during the selection process of applications are: (1) the need to have at least two implementations of the same application in different programming frameworks, or benchmark suites, and (2) applications that are compilable in our systems. Table 3 lists the considered applications from the SPEC Accel and Rodinia benchmark suite used for performance comparison of the selected parallel programming frameworks.

BFS, CFD, HotSpot, LUD, and NW are implemented using OpenCL, OpenMP and CUDA. The OpenCL implementation is provided by both SPEC Accel and Rodinia benchmark suites. No OpenACC implementation is provided by SPEC Accel for these applications. B+Tree, GE, HW, Kmeans, LavaMD, and SRAD are implemented using OpenCL and CUDA. While the OpenCL implementation is provided by both benchmarks, the CUDA implementation is provided by Rodinia. BP, k-NN, Leukocyte, Myocyte, PathFinder, PF, and SC are implemented using OpenCL and CUDA, and their implementations are provided by the Rodinia benchmark suite.

Additional information and implementation details for each of the considered benchmark applications are available at the documentation web-pages of SPEC Accel [16] and Rodinia [15].

3.2 Evaluation Metrics

In this section, we discuss the evaluation metrics considered for comparison of the selected parallel programming frameworks, including the required programming effort, the performance, and the energy efficiency.

3.2.1 Programming Productivity. To quantitatively evaluate the programming effort required to parallelize a program, we use our tool named *CodeStat* (see Section 2.1). We use *CodeStat* to determine the total lines of code LOC_{total} and the fraction of lines of code LOC_{par} written in OpenCL, OpenACC, OpenMP, or CUDA for a given application. We define the parallelization effort as follows,

Table 3: The considered applications from the SPEC Accel and the Rodinia benchmark suites.

		SPEC Accel		Rodinia		
Application	Domain	OpenCL	OpenACC	OpenMP	OpenCL	CUDA
LBM	Fluid Dynamics	x	x			
MRI-Q	Medicine	x	x			
Stencil	Thermodynamics	x	x			
BFS	Graph Algorithms	x		x	x	x
CFD	Fluid Dynamics	x		x	x	x
HotSpot	Physics Simulation	x		x	x	x
LUD	Linear Algebra	x		x	x	x
NW	Bioinformatics	x			x	x
B+Tree	Search	x			x	x
GE	Linear Algebra	x			x	x
Heartwall	Medical Imaging	x			x	x
Kmeans	Data Mining	x			x	x
LavaMD	Molecular Dynamics	x			x	x
SRAD	Image Processing	x			x	x
BP	Pattern Recognition				x	x
k-NN	Data Mining				x	x
Myocyte	Biological Simulation				x	x
PF	Medical Imaging				x	x
SC	Data Mining				x	x

$$Effort_{par}[\%] = 100 * LOC_{par}/LOC_{total} \quad (1)$$

3.2.2 Performance and Energy Consumption. We use the execution time (T) and the Energy (E) to express the performance and energy consumption. T is defined as the total amount of time that an application needs from the start till the end of the execution, whereas E is defined as the total amount of energy consumed by the system (including host CPUs and accelerators) from the beginning until the end of the execution.

The data for the execution time and the energy consumption are collected using the x-MeterPU tool (see Section 2.2). To collect such data, a wrapper class file was created. The control flow of this class is as follows: (1) start the time and energy counters; (2) synchronously execute the benchmark commands; and (3) stop the time and energy counters and calculate the total execution time and system energy consumption.

3.3 Results

In this section, we compare OpenMP, OpenACC, OpenCL, and CUDA with respect to (1) programming productivity, and (2) performance and energy consumption.

3.3.1 Programming Productivity. Table 4 shows the parallelization effort as percentage of code lines written in OpenCL, OpenACC, OpenMP, or CUDA that are required to parallelize various applications of Rodinia and SPEC Accel benchmark suites. We use Equation 1 in Section 3.2.1 to calculate the percentage of code lines.

Result 1: Programming with OpenCL requires significantly more effort than programming with OpenACC for the SPEC Accel benchmark suite.

Based on the available OpenCL and OpenACC code for *LBM, MRI-Q*, and *Stencil* from the SPEC Accel benchmark suite, we observe that on average OpenACC requires about 6.7× less programming effort compared to OpenCL.

Result 2: Programming with OpenCL on average requires about two times more effort than programming with CUDA for the Rodinia benchmark suite.

Table 4: Programming effort required to parallelize the code is expressed as percentage (see Equation 1) of code lines written in OpenCL, OpenACC, OpenMP, or CUDA. The remaining code lines are written in general-purpose C/C++.

	SPEC Accel		Rodinia		
	OpenCL[%]	OpenACC[%]	OpenMP[%]	OpenCL[%]	CUDA[%]
LBM	3.21	0.87			
MRI-Q	5.70	0.64			
Stencil	4.70	0.61			
BFS	6.95		4.86	9.07	12.50
CFD	5.83		2.53	9.00	8.08
HotSpot	4.75		2.67	13.18	8.20
LUD	5.78		2.30	9.72	7.82
NW	6.56			18.34	8.85
B+Tree	4.89			6.79	4.51
GE	9.63			14.21	9.76
Heartwall	5.34			6.74	3.97
Kmeans	2.80			2.67	2.17
LavaMD	4.61			9.24	7.74
SRAD	7.81			13.00	10.28
BP				12.21	5.95
k-NN				15.83	5.07
Myocyte				8.25	1.21
PF				17.83	9.47
SC				5.81	2.66

With respect to the comparison between OpenCL and CUDA using the applications in the Rodinia benchmark suite, except of the BFS implementation, on average CUDA requires 2× less programming effort than OpenCL.

Result 3: Programming with OpenMP requires less effort than programming with OpenCL and CUDA.

Based on the data collected for OpenMP, OpenCL, and CUDA implementations of *BFS, CFD, HotSpot*, and *LUD* from the Rodinia benchmark suite, we observe that on average OpenMP requires 3.6× less programming effort compared to OpenCL, and about 3.1× less programming effort compared to CUDA.

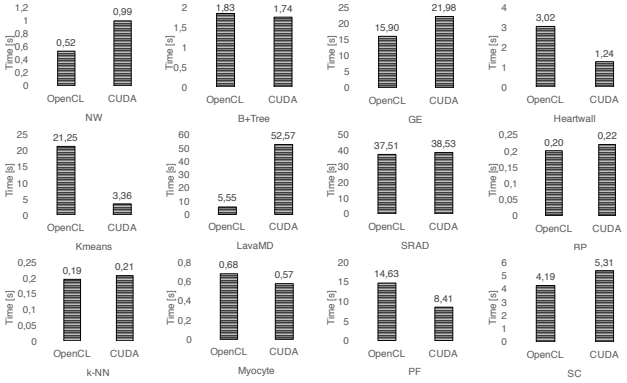
Result 4: The human factor can impact the fraction of code lines to parallelize the code.

We observe that the human factor significantly impacts the fraction of lines of code used to parallelize the code. For example, the OpenCL implementation of BFS on the SPEC Accel benchmark suite comprise 6.95% OpenCL specific lines of code, whereas the implementation on the Rodinia comprise 9.07% OpenCL specific lines of code. Differences in the required programming effort can be observed also for *CFD, HotSpot, LUD, NW, B+Tree, GE, Heartwall, Kmeans, LavaMD*, and *SRAD*.

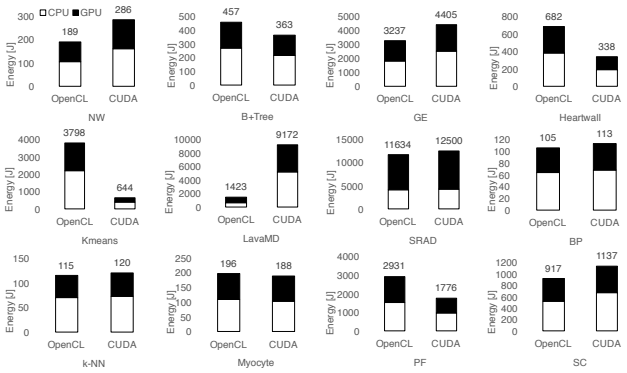
3.3.2 Performance and Energy. Figures 2, 3, and 4 depict the execution times and energy consumption for various applications of SPEC Accel and Rodinia benchmark suites on Emil and Ida.

Result 5: The performance and energy consumption behavior of OpenCL and CUDA are application dependent. For the Rodinia benchmark suite, for some applications OpenCL performs better, however there are several applications where CUDA performs better.

Figure 2 depicts the execution time and energy consumption of the OpenCL and CUDA implementations of *NW, B+Tree, GE, Heartwall, Kmeans, LavaMD, SRAD, BP, k-NN, Myocyte, PF*, and *SC* from the Rodinia benchmark suite on the GPU-accelerated system *Ida*. Results show that the OpenCL implementation of seven out of 12 applications, including *NW, GE, LavaMD, SRAD, BP, k-NN*, and *SC* perform better than their corresponding CUDA implementations.

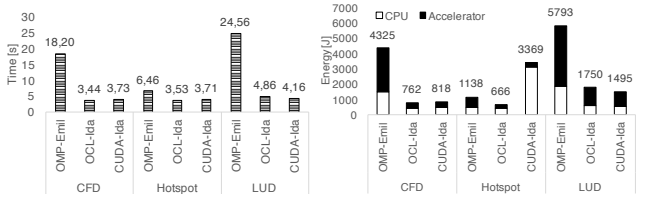


(a) Time



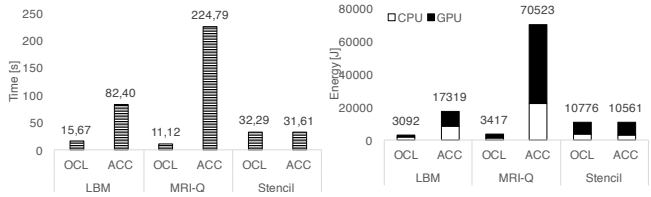
(b) Energy Consumption

Figure 2: A comparison of OpenCL and CUDA with respect to (a) execution time and (b) energy consumption. Results are obtained for various applications from the Rodinia benchmark suite (Table 3) on the GPU-accelerated system *Ida* (Table 2).



(a) Time **(b) Energy Consumption**

Figure 3: A comparison of OpenMP, OpenCL, and CUDA with respect to (a) execution time and (b) energy consumption using the Rodinia benchmark suite (Table 3). OpenMP (OMP) versions of CFD, HotSpot, LUD are executed on the Intel Xeon Phi accelerated system *Emil* (Table 2), whereas OpenCL (OCL) and CUDA versions of CFD, HotSpot, LUD are executed on the GPU-accelerated system *Ida*.



(a) Time **(b) Energy Consumption**

Figure 4: A comparison of OpenCL and OpenACC with respect to (a) execution time and (b) energy consumption. Results are obtained using LBM, MRI-Q, and Stencil from the SPEC Accel benchmark suite (Table 3) on the GPU-accelerated system *Ida* (Table 2).

While for most of the applications, including *B+Tree, GE, SRAD, BP, k-NN, Myocyte*, and *SC*, the performance of OpenCL and CUDA is comparable, for some applications such as *NW* and *LavaMD* there is a large performance difference where OpenCL performs better than CUDA, whereas for *Heartwall, Kmeans,* and *PF* the CUDA implementations perform better than their OpenCL counterparts do.

Result 6: Less execution time results with less energy consumption.

Figure 2 shows that, for all implementations of application benchmarks, those that had lower execution time had lower energy consumption. For instance, the OpenCL implementation of NW is faster and consumes less energy than the corresponding CUDA implementation.

Result 7: OpenMP implementations of CFD, HotSpot, and LUD executed on Emil, perform significantly slower than the corresponding OpenCL and CUDA implementations executed on Ida.

Figure 3 depicts the comparison of OpenMP, OpenCL, and CUDA with respect to the execution time and energy consumption using the Rodinia benchmark suite. Figure 3a shows the execution time of *CFD, HotSpot*, and *LUD* applications. The OpenMP implementation of these applications is executed on the Intel Xeon Phi accelerated system *Emil*, whereas the OpenCL and CUDA versions are executed on the GPU accelerated system *Ida*. We may observe that the OpenCL and CUDA versions execution time are comparable, whereas the OpenMP implementation is significantly slower. Please note that the host CPUs of *Emil* are Intel Xeon E5-2695 v2, whereas the host CPUs of *Ida* are of type E5-2650 v4. Furthermore, the Intel Xeon Phi 7120P co-processor on *Emil* is the first generation of the Intel Xeon Phi architecture (known as Knights Corner).

Result 8: OpenCL performs better than OpenACC for the SPEC Accel benchmark suite on Ida.

Figure 4 depicts the comparison of OpenCL and OpenACC with respect to execution time and energy consumption using *LBM, MRI-Q,* and *Stencil* from the SPEC Accel benchmark suite on the GPU-accelerated system *Ida*. We may observe that the OpenCL implementation of LBM and MRI-Q is significantly faster than the corresponding OpenACC implementation, whereas for the implementations of Stencil the results are comparable. However, according to the results showed in Table 4 writing code for OpenCL demands significantly greater effort than writing OpenACC code.

4 RELATED WORK

In this section, we highlight examples of research that addresses programming aspects of heterogeneous computing systems and position this paper with respect to the related work.

Su *et al.* [18] compare the performance of OpenCL and CUDA using five kernels: Sobel filter, Gaussian filter, median filter, motion estimation, and disparity estimation. The execution time of CUDA-based implementations was 3.8% – 5.4% faster than the OpenCL-based implementations.

Kessler *et al.* [7] study three approaches for programmability and performance portability for heterogeneous computing systems: SkePU skeleton programming library, StarPU runtime system, and Offload C++ language extension. Authors reason about the advantages of each of these approaches and propose how they could be integrated together in the context of the PEPPHER project [2] to achieve programmability and performance portability.

Mittal and Vetter [11] provide a comprehensive coverage of literature for GPU-accelerated computing systems. In this context, the authors survey the literature about runtime systems, algorithms, programming languages, compilers, and applications.

Li *et al.* [8] study empirically the programmer productivity in the context of OpenACC with CUDA. The study that involved 28 students at undergraduate and graduate level was performed at Auburn University in 2016. The students received the sequential code of two programs and they had to parallelize them using both CUDA and OpenACC. From 28 students, only a fraction was able to complete properly the assignments using CUDA and OpenACC. The authors conclude that OpenACC enables programmers to shorten the program development time compared to CUDA. However, the programs developed with CUDA execute faster than their OpenACC counterparts do.

This paper complements the related research with an empirical study of four widely used frameworks for programming heterogeneous systems: OpenCL, OpenACC, OpenMP, and CUDA. Using de-facto standard benchmark suites SPEC and Rodinia we study the productivity, performance, and energy consumption. Our measurement tools CodeStat and x-MeterPU that we developed for this study enable us to address systems accelerated with Intel Xeon Phi and GPU.

5 SUMMARY

We have presented a study of productivity, performance, and energy for OpenMP, OpenACC, OpenCL, and CUDA. For comparison we used the SPEC Accel and Rodinia benchmark suites on two heterogeneous computing systems: *Emil* that comprises two Intel Xeon E5 processors and one Intel Xeon Phi co-processor, and *Ida* that has two Intel Xeon E5 processors and one GTX Titan X GPU. Our major observations include: (1) programming with OpenCL requires significantly more effort than programming with OpenACC for SPEC Accel benchmark suite; (2) programming with OpenCL on average requires about two times more effort than programming with CUDA for Rodinia benchmark suite; (3) the human factor can impact the fraction of code lines to parallelize the code; (4) less execution time results with less energy consumption; and (5) OpenCL performs better than OpenACC for SPEC Accel benchmark suite on Ida.

Future work will address new architectures of Intel Xeon Phi and NVIDIA GPU. Beside Rodinia and SPEC benchmarks we plan to use real-world applications.

REFERENCES

[1] Erika Abraham, Costas Bekas, Ivona Brandic, Samir Genaim, Einar Broch Johnsen, Ivan Kondov, Sabri Pllana, and A. Achim Streit. 2015. Preparing HPC Applications for Exascale: Challenges and Recommendations. In *18th International Conference on Network-Based Information Systems (NBiS)*. 401–406. https://doi.org/10.1109/NBiS.2015.61

[2] Siegfried Benkner, Sabri Pllana, Jesper Larsson Traff, Philippas Tsigas, Uwe Dolinsky, Cedric Augonnet, Beverly Bachmayer, Christoph Kessler, David Moloney, and Vitaly Osipov. 2011. PEPPHER: Efficient and Productive Usage of Hybrid Computing Systems. *Micro, IEEE* 31, 5 (Sept 2011), 28–41.

[3] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC 2009*. IEEE, 44–54.

[4] George Chrysos. 2014. Intel® Xeon Phi™ Coprocessor-the Architecture. *Intel Whitepaper* (2014).

[5] Daniel Grzonka, Agnieszka Jakobik, Joanna Kolodziej, and Sabri Pllana. 2017. Using a multi-agent system and artificial intelligence for monitoring and improving the cloud performance and security. *Future Generation Computer Systems* (2017). https://doi.org/10.1016/j.future.2017.05.046

[6] Guido Juckeland, William Brantley, Sunita Chandrasekaran, Barbara Chapman, Shuai Che, Mathew Colgrove, Huiyu Feng, Alexander Grund, Robert Henschel, Wen-Mei W Hwu, et al. 2014. SPEC ACCEL: a standard application suite for measuring hardware accelerator performance. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 46–67.

[7] Christoph Kessler, Usman Dastgeer, Samuel Thibault, Raymond Namyst, Andrew Richards, Uwe Dolinsky, Siegfried Benkner, Jesper Larsson Traff, and Sabri Pllana. 2012. Programmability and performance portability aspects of heterogeneous multi-/manycore systems. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1403–1408. https://doi.org/10.1109/DATE.2012.6176582

[8] Xuechao Li, Po-Chou Shih, Jeffrey Overbey, Cheryl Seals, and Alvin Lim. 2016. Comparing programmer productivity in OpenACC and CUDA: an empirical investigation. *International Journal of Computer Science, Engineering and Applications (IJCSEA)* 6, 5 (2016), 1–15. https://doi.org/10.5121/ijcsea.2016.6501

[9] Lu Li and Christoph Kessler. 2016. MeterPU: A Generic Measurement Abstraction API Enabling Energy-tuned Skeleton Backend Selection. *Journal of Supercomputing* (2016), 1–16. https://doi.org/10.1007/s11227-016-1792-x

[10] Suejb Memeti and Sabri Pllana. 2015. Accelerating DNA Sequence Analysis Using Intel(R) Xeon Phi(TM). In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 3. 222–227.

[11] Sparsh Mittal and Jeffrey S Vetter. 2015. A survey of cpu-gpu heterogeneous computing techniques. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 69.

[12] NVIDIA. 2016. CUDA C Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/. (September 2016). Accessed: 2017-03-06.

[13] NVIDIA. 2017. What is GPU-Accelerated Computing? http://www.nvidia.com/object/what-is-gpu-computing.html. (April 2017). Accessed: 2017-04-03.

[14] OpenMP. 2013. OpenMP 4.0 Specifications. http://www.openmp.org/specifications/. (July 2013). Accessed: 2017-03-10.

[15] Rodinia. 2015. Rodinia:Accelerating Compute-Intensive Applications with Accelerators. (December 2015). http://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Rodinia:Accelerating_Compute-Intensive_Applications_with_Accelerators Last accessed: 10 April 2017.

[16] SPEC. 2017. SPEC ACCEL: Read Me First. https://www.spec.org/accel/docs/readme1st.html#Q11. (February 2017). Accessed: 2017-04-10.

[17] John E Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering* 12, 1-3 (2010), 66–73.

[18] Ching-Lung Su, Po-Yu Chen, Chun-Chieh Lan, Long-Sheng Huang, and Kuo-Hsuan Wu. 2012. Overview and comparison of OpenCL and CUDA technology for GPGPU. In *2012 IEEE Asia Pacific Conference on Circuits and Systems*. 448–451. https://doi.org/10.1109/APCCAS.2012.6419068

[19] Andre Viebke and Sabri Pllana. 2015. The Potential of the Intel (R) Xeon Phi for Supervised Deep Learning. In *2015 IEEE 17th International Conference on High Performance Computing and Communications*. 758–765. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.45

[20] Sandra Wienke, Paul Springer, Christian Terboven, and Dieter an Mey. 2012. OpenACC: First Experiences with Real-world Applications. In *Proceedings of the 18th International Conference on Parallel Processing (Euro-Par'12)*. Springer-Verlag, Berlin, Heidelberg, 859–870.

[21] Yonghong Yan, Barbara M. Chapman, and Michael Wong. 2015. A comparison of heterogeneous and manycore programming models. https://goo.gl/81A4iV. (March 2015). Accessed: 2017-03-31.